CERN

Java Workshop on COMSOL FEM Automated Workflow

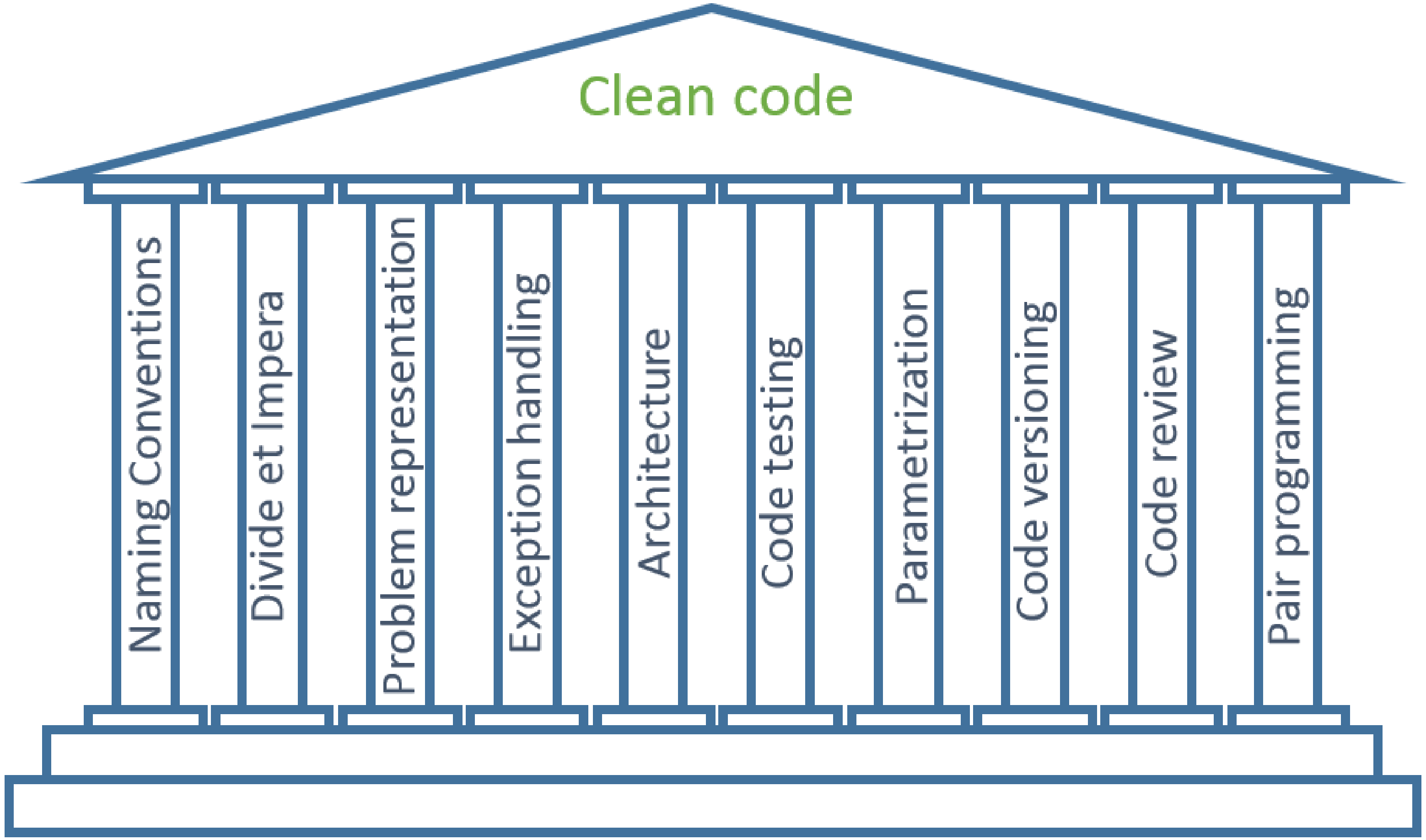
THE STEAM

CERN

Outline

- STEAM conventions
- FEM workflow
- Magnet as a composition of physical domains
- The example of coil domain
  - Identification of handles for Geometry and Properties
- Proposal for an API-free low level

STEAM conventions - good coding practices



9 July 2015, Clean code development workshop, jointly with MPE/MS
13 Aug 2015, Object oriented programming workshop, jointly with MPE/MS

GitLab

Code Repository
Code Review
Continuous Integration

sonarqube

Static code analysis

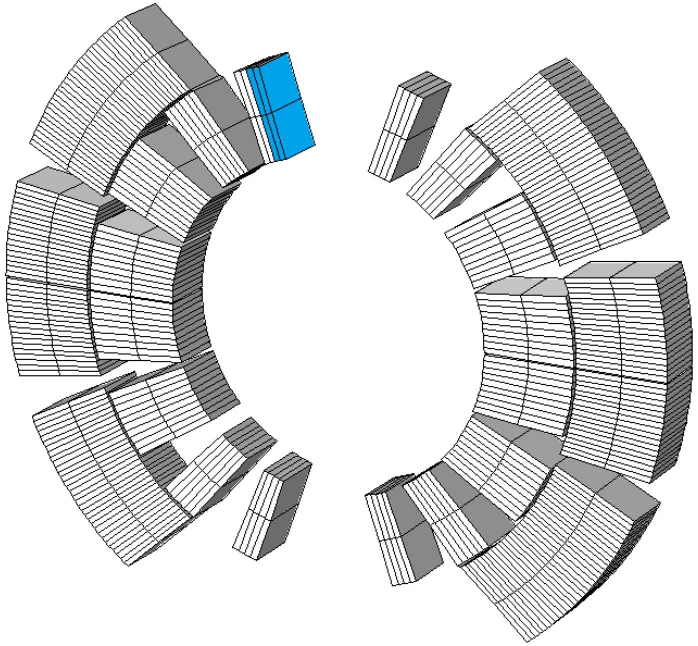
IntelliJ IDEA

Java IDE

Daily stand-up meetings @10AM

CERN

FEM workflow



- 320 domains (cable cross sections)
- Domain-dependent equations
- GUI workflow risky, slow, error-prone
- **Automation as solution**

User's Input
Magnet features

TXT

Numerical Engine

COMSOL API

Java

model

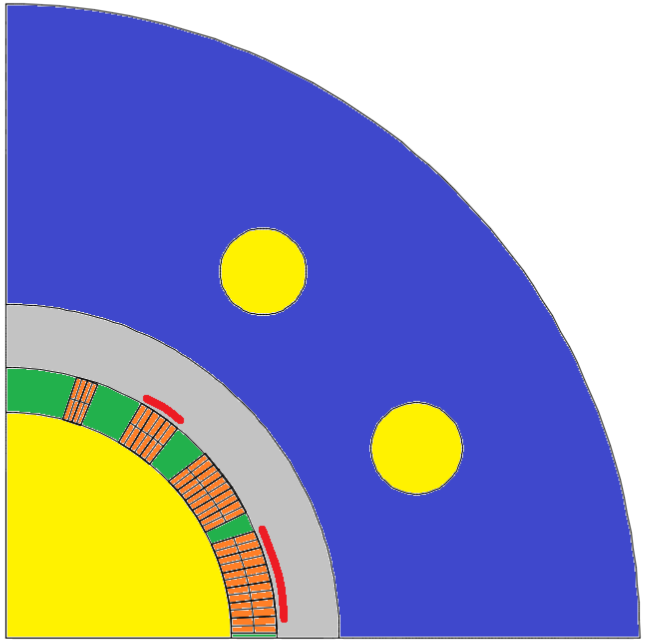
C-functions
Material Properties

{C} PROGRAMMING

Solver

Post-processing

Magnet - Natural composition of Domains



Air
Iron yoke
Steel collar
Quench heaters
Coil
Structural wedges
…

Each Domain has the following properties:

- **Geometry**
  e.g. points, lines, surfaces, volumes

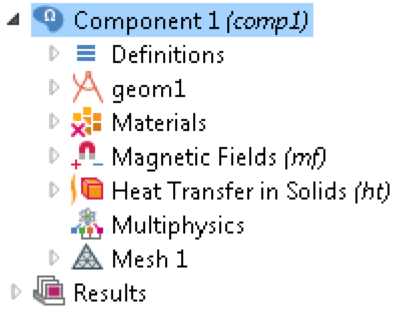
- **Material**
  e.g. copper, iron, polymide

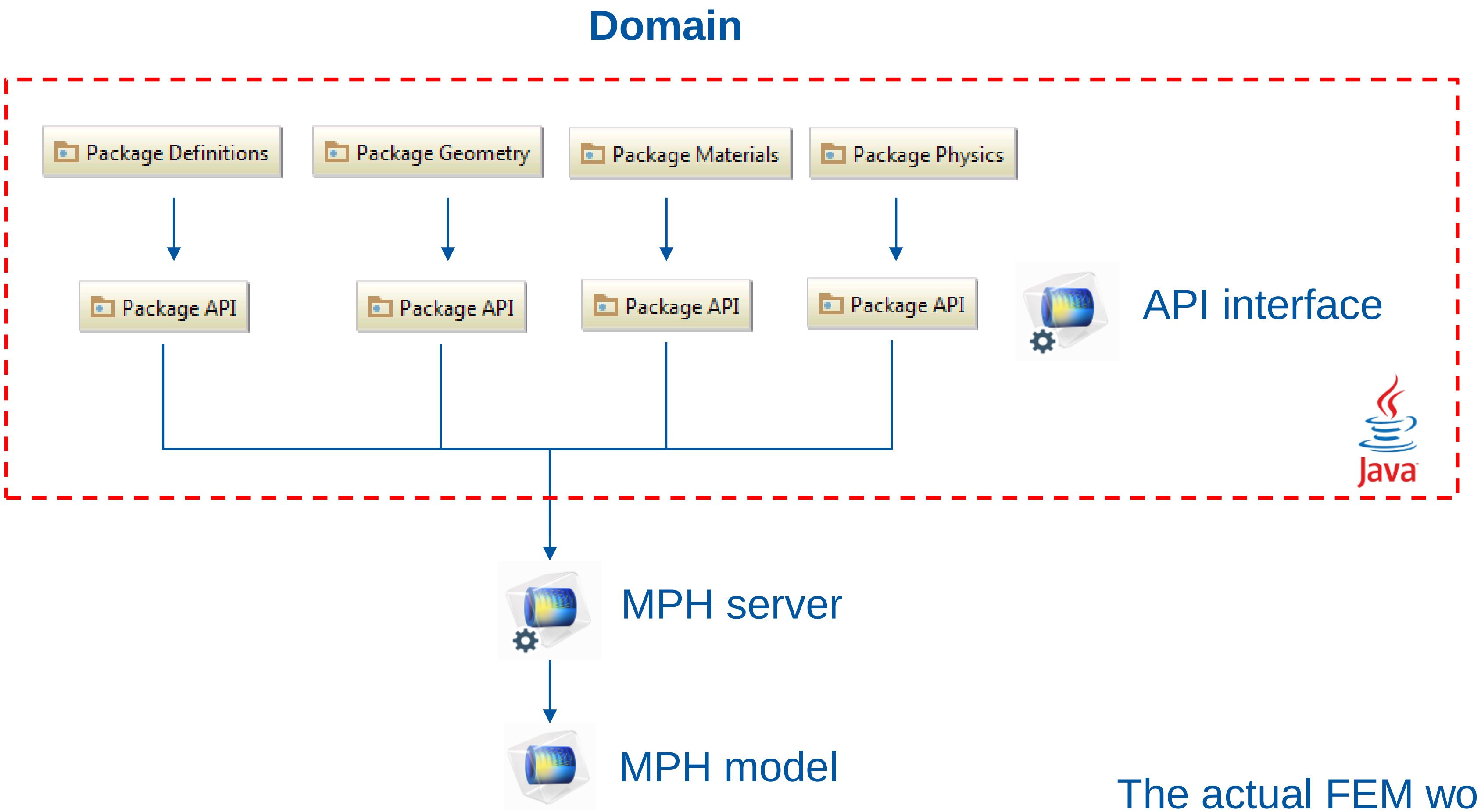
- **Physics**
  e.g. Ampere's law, External Current Density, Heat souce

IF every domain is assigned with these three properties, the model can be processed with a FEM tool.

Domain representation in Java

The architecture is inspired by the COMSOL one

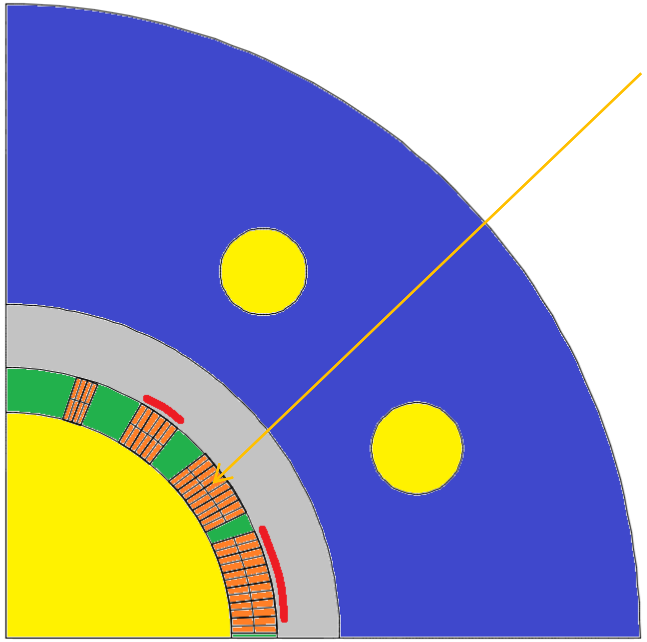




The actual FEM workflow is tightly coupled to COMSOL

The example of coil domain

Domain: A composition of **elements**, witch **differ** in geometry but **share** the same Material and Physics.



Coil domain

- Unique label
- Made by a *Material*
- Implements some *Physics Laws*
- Contains the element *Coil (1)*

Properties

Geometry

- *Coil* is made of *Windings (4)*
    - *Windings* is made of *HalfTurns (6,4,3,2)*
        - *HalfTurns* is made of *Polygons (6,4,3,2)*

Bottom Line: *Polygon* is an elementary geometrical concept

Geometry API: The polygon example

Definition

Plane, convex figure that is bounded by a finite chain of straight line segments closing in a loop

Consequences

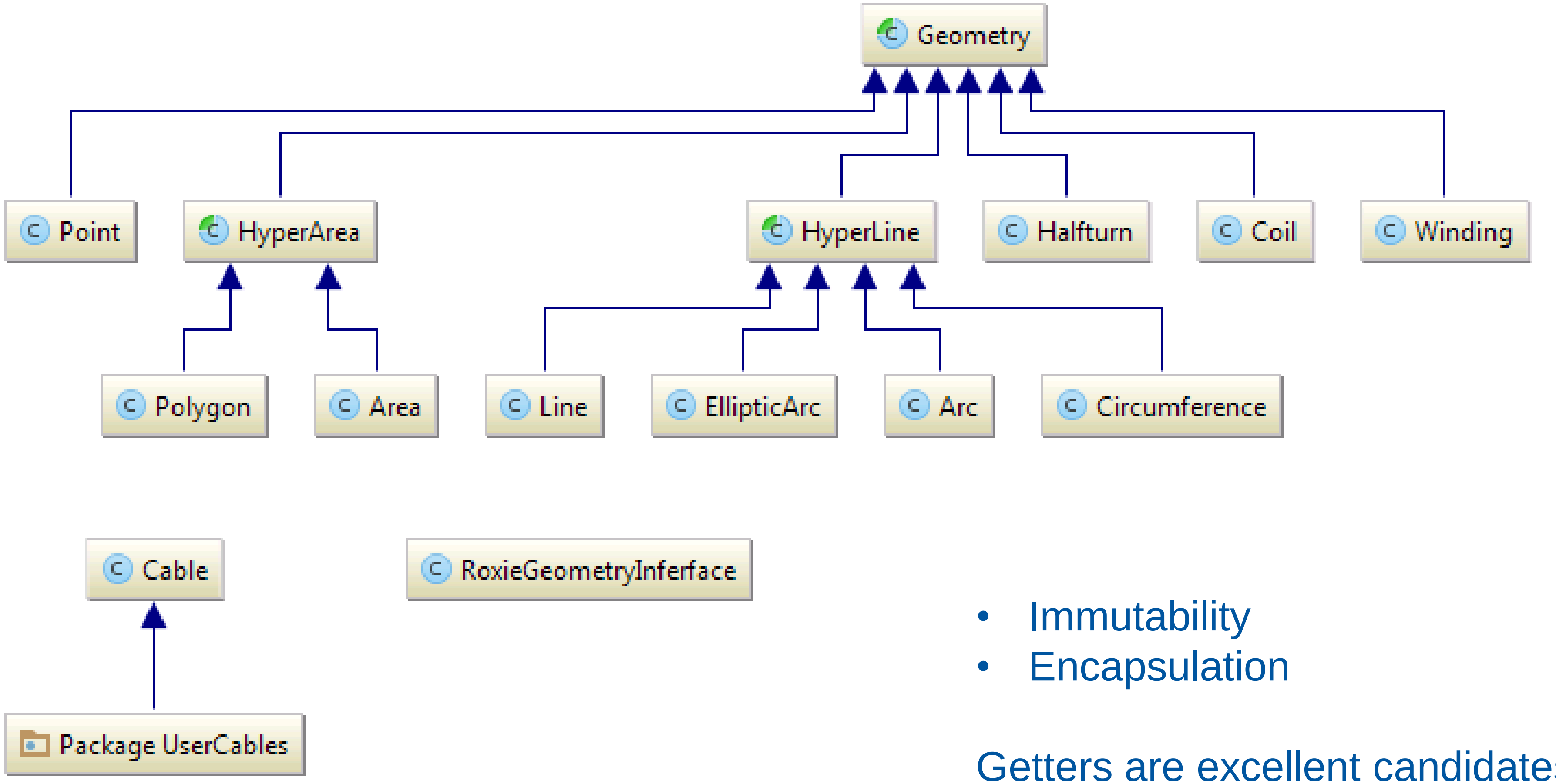
- Indipendent from any code or API
- API reflects the definition, so polygon is a general handle for any low-level implementation

```
public void createPolygon(Model mph, String label, Polygon poly) {
    String localFeatureLabel = "Polygon";
    mph.geom(geomName).create(label, localFeatureLabel);
    mph.geom(geomName).feature(label).label(label);
    mph.geom(geomName).feature(label).set("source", "table");

    Point[] parray = poly.extractVertices();

    for (int point_i = 0; point_i<parray.length; point_i++) {
        mph.geom(geomName).feature(label).setIndex("table", parray[point_i].getX() + "", point_i, 0);
        mph.geom(geomName).feature(label).setIndex("table", parray[point_i].getY() + "", point_i, 1);
    }
    mph.geom(geomName).feature(label).set("selresult", "on");
    mph.geom(geomName).feature(label).set("selresultshow", "all");
}
```

Geometry UML Class Diagram



- Immutability
- Encapsulation

Getters are excellent candidates as handles!

Handles for the remaining properties

- Geometry helps in defining general java objects (e.g. a circle) that can be associeted to a handle
- This is not the case for the remaining properties, e.g. Materials and Physics Laws.
- The coding of Materials and Physics Laws is widely tailored on the requirements from COMSOL API that requires mostly strings.

```
public void addNodeExternalCurrentDensity(Model mph, String nodeLabel, String[] J_xyz){
    mph.physics(physName).create(nodeLabel, "ExternalCurrentDensity", 2);
    mph.physics(physName).feature(nodeLabel).label(nodeLabel);
    mph.physics(physName).feature(nodeLabel).set("Je", J_xyz);
}
```

```
public void addCp(Model mph, String label, String value) {
    mph.material(label).propertyGroup("def").set("heatcapacity", value);
}
```

The code converts the user’s input (e.g. external current density) in a set of strings, used as arguments in the related API implementation.

Proposal for an API-free architecture

Real-time model building

Use input

Model Builder

Java

Offline model building

Strings

Strings

API interface

↓

MPH server

↓

MPH model

MACRO that tabulates the API instructions on an external file

↓

Interpreter/compiler

TXT

GetDP

ANSYS

Example of Implementation

... From an API-based low level

```
public void addCp(Model mph, String label, String value) {
    mph.material(label).propertyGroup("def").set("heatcapacity", value);
}
```

... To a String-based Low level

```
public void addCp(Model mph, String label, String value) {
String commandLine = String.format("model.material(%s).propertyGroup(\"def\").set(\"heatcapacity\",%s)",name, value);
ExternalFile.NewLine.Write(commandLine)
}
```

Instructions are sent to a .txt file that can be compiled later by the FEM solver (.txt can be verisoned, maintainability of the library of models)

```
model.material("Copper").propertyGroup("def").set("heatcapacity", 500[J/kg/K] )
```

www.cern.ch